Research in Computational Astrobiology

Final Technical Report for Contract NCC2-5389

Silvano Colombano[a], Greg Laughlin[b], Andrew Pohorille[c,d], and Michael A. Wilson[c,d,e]

[a]MS 230-3, NASA Ames Research Center, Moffett Field, CA 94035

[b]UCO/Lick Observatory, University of California, Santa Cruz, 95064

[a]MS 239-4, NASA Ames Research Center, Moffett Field, CA 94035

[d]Dept. Pharmaceutical Chemistry, University of California, San Francisco, 94143

[e]Principal Investigator

Period of Performance: July 1, 2000 – June 30, 2001

Inventions: There are no patents or inventions arising from this research project.

Abstract

We have pursued several projects in the new field of computational astrobiology, which is devoted to advancing our understanding of the origin, evolution and distribution of life in the Universe using theoretical and computational tools. We have developed an algorithm for calculating long-range effects in molecular dynamics using a multipole expansion of the electrostatic potential. This algorithm is expected to be highly efficient for simulating biological systems on massively parallel supercomputers. Work has been done to establish the NASA Resource in Bioinformatics. This resource will support studies on the origins and evolution of life, and space genomics (including the effects of gravity on gene expression). In order to understand better the self-organization of organic matter in the earliest cells, and its evolution, we will develop models of protobiological chemistry using neural networks. Finally, we have performed simulations of three planet systems to investigate models of planetary system formation that might account for recently discovered extrasolar planetary systems.

Introduction

The goal of this research was to pursue research in computational astrobiology through a series of projects which address several issues identified in the recent NASA Astrobiology Roadmap. Computational Astrobiology uses computational and theoretical techniques to advance our understanding of the origin, evolution and distribution of life in the Universe. Computational astrobiology approaches these problems from several different points of view simultaneously, ranging from the molecular and cellular level to the ecological and biosphere level. This requires exploiting information from not only the biological sciences, but also chemistry, geology, paleontology, and planetary and atmospheric sciences. Similarly, the

goals of computational astrobiology cannot be accomplished using a single area of computer science but, instead, involve creative integration of several traditionally separate disciplines: biomodeling and biosimulations, bioinformatics, and complex systems science. This is exactly the approach taken in this proposal. Below we describe several projects that were carried out by several undergraduate and graduate students in collaboration with scientists at NASA Ames.

Long-range Effects in Molecular Dynamics. To assist Ames' efforts in developing the fastest code for molecular-level simulation of biological systems on massively parallel supercomputers, we have developed a modified, highly efficient, state-of-the-art, multipole expansion code to treat long range effects in molecular dynamics. The newly developed code will be applied to the study of two outstanding problems in astrobiology: (i) understanding the structure and mechanism of action of the first proteins evolved from random sequences by in vitro selection; and (ii) designing simple membrane proteins capable of transporting material across cell walls, utilizing energy captured from the environment and transmitting signals from the environment into the prebiotic cell. This work will truly advance both the state-of-the-art of research in astrobiology and the connection between astrobiology and information technology at Ames.

Establishing NASA Resource in Bioinformatics. We have helped establish a bioinformatics infrastructure at Ames in order to support current and future biological research at NASA. Such research includes studies on the origins and evolution of life, and space genomics (including the effects of gravity on gene expression). It will provide a mechanism for NASA researchers and their collaborators to integrate NASA proprietary genomic data with other, rapidly growing databases, and to organize, view and analyze these data in a way that is specific to individual research goals.

NASA's interests in bioinformatics center around questions regarding how critical cellular functions evolved, were selected for, and respond to conditions in space. This will help, for example, to identify minimum requirements for life, which can be further used as a criterion in the search for life on other planets. Another important research goal is to explain how genes are regulated and respond to the absence of gravity. Bioinformatics related research and infrastructure are essential to Astrobiology, Fundamental Biology and Space Genetics Programs at NASA. It will also be an inherent component of analysis of data from all future biology-oriented NASA missions. Thus, the formation of the Bioinformatics Resource is critical for the success of these programs and missions.

We have assisted Ames in establishing core competency in bioinformatics by obtaining, testing, and deploying analysis and database software for use by NASA researchers. Some of the software programs required are: CLUSTALW (multiple alignment), Phylip (phylogenetic analysis), SAM, Hmmer, HmmPro (Hidden Markov model building and analysis), MEME, Mast (motif search), BLAST2, WU-BLAST (homology detection), Modeller (homology modeling), Rasmol (molecular visualization), and various analysis software developed in house. Access to the software will be web-based through a secure server on a centralized computer system. This would set the foundation for a NASA repository of biological information,

which could be integrated with already available public data.

Protobiological Chemistries using MolNets. In general, artificial neural networks can be viewed as structures that perform specific functions. Given any particular topology and connection weights, input signals will be transformed into specific output signals. This input-output relationship is the "function"' performed by the neural structure and is uniquely determined by this structure and by the input (environment).

For our purposes, the traditional neural network idea will be modified as follows: inputs to the network will not be simply transformed into output signals but will be also utilized to modify the topology of the network. The rules for these transformations and modifications are very simple and constitute the "physics" of the system. The essence of this model is that the networks are placed in a spatial dimension, come into contact with one another and modify their structures as a result of these interactions. They act like like molecules in a pseudo-chemical soup, hence the name "MolNets" e.g. MolNets A and B can meet and produce MolNets C and D.

Dynamical Interactions within Three-Planet Systems. The number of habitable planets in the galaxy is a quantity of fundamental astrobiological interest. In the past five years, over 80 extrasolar planets have been detected, which are presenting us with some major surprises. In particualr, none of the planetary systems discovered thus far bears much resemblance to our own Solar System. In some cases, we are finding Jupiter-mass planets at very small distances from the parent star. In other cases, we are finding massive planets with very eccentric orbits, in stark contrast to the near-circular orbits of the major planets in our system.

We have performed a large-scale set of calculations geared toward answering the question: Is it possible that the currently observed census of eccentric, short-period Jupiter-mass planets arises from catastrophic gravitational interactions among a parent population of young Jupitermass planets which formed in circular orbits at distances greater than five astronomical units from the parent star? If so, the present model for planetary system formation can easily incorporate the diverse population of planets now known.

Results

1 Long-range Effects in Molecular Dynamics

(This project was carried out by Kevin Lin, a graduate student in mathematics from U.C. Berkeley with Andrew Pohorille and Michael Wilson)

Implementation of the Fast Multipole Method

Kevin K. Lin
kkylin@math.berkeley.edu

Abstract

The aim of this document is to record most of the nitty-gritty details of my implementation of the adaptive fast multipole method: Everything from the overall structure to specific recurrence relations, as well as obvious deficiencies and fundamental limitations.

1.1 Introduction

This article describes my implementation of the fast multipole method (FMM) during the summer of 2000.[1] A quick description of the adaptive fast multipole method and modifications necessary for periodic boundary conditions is followed by a detailed discussion of the internals of the source code: Representations of data structures as Fortran 77 arrays, obvious inefficiencies, and fundamental limitations of these algorithms. This article should serve two purposes: First, to review the basic principles of the fast multipole method and to send interested readers to appropriate references, and second, to document the nuts and bolts of my Fortran implementation. This is only a draft, so it is bound to be incomplete and I welcome all comments, questions, and suggestions.

A note on the source code. The souce code is basically written in plain-vanilla Fortran 77, although some parts rely heavily on SGI Fortran extensions.[2] The code was deliberately written to mimic the mathematics and thus may seem intentionally slow in some ways. Most of these inefficiencies, however, do not involve fundamental data structures and are, I hope, easy to spot and modify.

1.1.1 Key references

The overall structure of the program follows Cheng, Greengard, and Rokhlin (Cheng *et al.*, 1999) very closely. In particular, interested readers who have not done so should take a look at the pseudocode in §4 of (Cheng *et al.*, 1999), which gives a very careful description of the adaptive fast multipole method.[3] The mechanism for computing forces with periodic boundary conditions follows Challacombe *et. al.* (Challacombe *et al.*, 1997): My code reproduces their results, but as far as I know their calculations have not been independently verified with Ewald or direct summation methods. There are mistakes in (Challacombe *et al.*, 1997), by the way: See Section 1.3 for details.

[1] This work was done as part of an internship at the Center for Computational Astrobiology at NASA's Ames Research Center.

[2] In particular, the program `fmm_ndw` relies heavily on extensions for quadruple precision arithmetic. (`fmm_ndw` computes lattice interaction matrices for the periodic FMM.)

[3] One major difference is that my implementation uses recurrence-based translation operators (discussed in (Lin, 2000)), which require a two-box separation for multipole-to-local translations. In contrast, Cheng *et. al.* require only a separation of one box. This affects the definitions of neighbor lists.

1.1.2 Notation

The notation in this article follows (Lin, 2000) and (White and Head-Gordon, 1994): Throughout, j, k, l, m are reserved as indices of multipole or local expansions, and p is the order of the expansion. $O_{l,m}$ always refers to the multipole coefficient for a *single charge of* $+1$, whereas $\hat{O}_{l,m}$ always refers to the multipole coefficient for a system of charges. Similarly, $M_{l,m}$ refer to local coefficients for a single charge, whereas $\hat{M}_{l,m}$ refer to a system of charges.

1.1.3 Document organization

Section 1.2 reviews the basic structure of the adaptive FMM and discusses differences between my implementation and that of (Cheng *et al.*, 1999). It also brings to light some of the rationale behind the algorithm and points out common pitfalls. Section 1.3 then discusses modifications necessary to use periodic boundary conditions efficiently. Section 1.4 covers the *exact* recurrence relations employed by the program to build tables of multipole and local expansions, as well as the details of how the translation operators are implemented, and Section 1.5 describes the detailed implementation of the FMM. Finally, Section 1.6 offers some suggestions on optimization, and Section 1.7 points out some fundamental (and some not-so-fundamental) limitations of the code.

Before proceeding, the reader may wish to consult (Lin, 2000) for a description of multipole and local expansions and the translation operators used to manipulate them. (What follows assumes familiarity with that material.)

1.2 The Adaptive FMM: Basic Structure

I implemented a variant of Cheng, Greengard, and Rokhlin's adaptive fast multipole method (Cheng *et al.*, 1999). The code is somewhat complicated because of the need to keep track of different types of neighboring boxes, but the presentation in (Cheng *et al.*, 1999) is quite good. Below, the adaptive algorithm is reviewed, and interested readers are referred to §4 of (Cheng *et al.*, 1999) for a more detailed discussion. Also, this discussion assumes familiarity with the basics of multipole and local expansions and their translation operators (see (Lin, 2000)). Throughout the discussion, I will make references to important parameters in the code, but a detailed discussion of the program itself is deferred until Section 1.5.

The fast multipole method begins by forming a box around the entire system of charges, then forming equal octants and distributing particles into each octant. This process is repeated for $\texttt{maxlevel} - 1$ times, until the smallest boxes have sides that are $2^{\texttt{maxlevel}-1}$ that of the largest box, where `maxlevel` is the maximum allowable *depth* of the tree. This process defines an *octree*. The non-adaptive multipole method defines a *full* octree: Each *level* of the tree has $8^{\texttt{level}-1}$ boxes (this means the tree contains a total of $\frac{1}{7}(8^{\texttt{level}} - 1)$ boxes). The largest box occupies level 1 by itself, and boxes in increasing levels are *smaller* in size and farther down the tree.[4] The adaptive method, in contrast, only divides a box into octants

[4] I know it's confusing to have increasing level correspond to decreasing size, but it's stuck. Sorry about that.

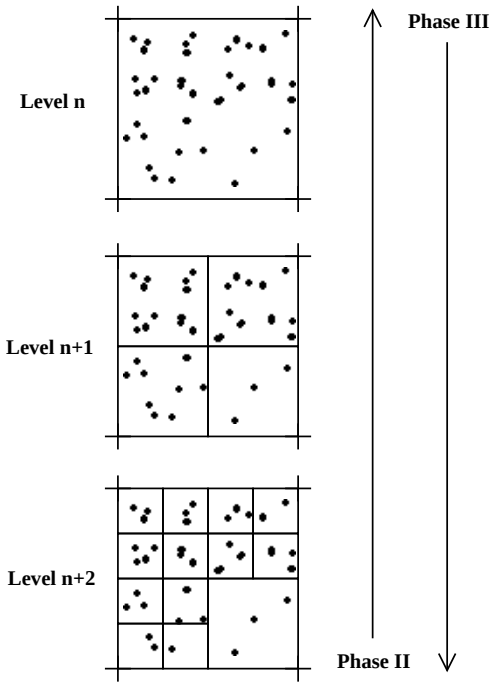

Figure 1: Adaptive construction of child boxes: This shows three levels of the tree. Note that this is a two dimensional system (but the three dimensional case is similar).

if the number of particles in that box exceeds a certain fixed number. This helps balance the number of particles per *childless box* and avoids wasting time on boxes which are nearly empty. As usual, if `b` denotes a box and `bc` denotes one of its octants, then `b` is `bc`'s *parent* and `bc` is one of `b`'s *children*. Boxes without children are said to be*childless* and corresponds to the tree's "leaves." Every box that is not childless has eight children.

Here is a very vague description of the fast multipole method:

Phase I: Load particles and build tree structure.

Phase II (bottom-up): Build multipole expansions for childless boxes, then propagate multipole expansions up the tree using multipole-to-multipole translation operators. This constructs multipole expansions for all boxes, which determines far-field forces generated by charges in each box.[5]

Phase III (top-down): Starting at the "top,"[6] for each pair of "sufficiently distant" boxes `b` and `bb`, construct a local expansion about the center of `b` for the field generated by `bb` by applying the multipole-to-local translation operator to the multipole expansion of `bb`. This local expansion can then be translated and propagated to `b`'s children.

[5] I will often write "forces generated by a box" when the more precise statement is "forces generated by charges in a box."

[6] For technical reasons, explained below, "top" can mean either level 1 or level 3, depending on whether one enforces periodic boundary conditions or not.

This recursively exchanges far-field effects between distant boxes, if the "sufficiently distant" boxes are defined correctly. For the adaptive method, each box also needs to evaluate the field generated by nearby large boxes.

Phase IV (evaluate forces): For each childless box, interact directly with nearest neighbors and evaluate local expansion to obtain forces on each charge. In the adaptive case, this stage should also make sure that forces generated by nearby small boxes are accounted for.

1.2.1 Phase I: Build tree structure

This section describes the implementation of the particle-loading phase:

Step 1: Load particles. Beginning with level 1 and going to the maximum level allowed (`maxlevel`), iteratively distribute particles into boxes. A given box `b` is divided into octants only if the number of particles in `b` exceeds a fixed limit, `maxcount`. (A simple running time analysis shows that, for optimal speed, `maxcount` should be proportational to $p^{3/2}$. The constant of proportionality is best found by experimentation.)

Step 2: Build neighbor lists. Every box has five lists (labeled 0 through 4) associated with it. These lists contain indices of nearby boxes for computing electrostatic interactions. Defining these lists can be somewhat complicated and is done in detail later. For now, it is only necessary to note that three of these lists (Lists 1, 3, and 4) contain nearby boxes that are too close for multipole-to-local conversion to converge quickly, and List 2 contains boxes that are just far enough away that multipole-to-local conversion works really well.

Step 3 (optional): Balance tree. For some particle distributions, the simple tree construction described above may result in suboptimal divisions: Boxes with just a few more particles than `maxcount` may get divided, for example, which results in child boxes that have too few particles. It may be useful to go through the tree and "prune" away childless boxes with too few particles, though the best way to do this is far from obvious.

1.2.2 Phase II

This section describes the "bottom-up" phase (see Figure 1.2):

Step 1: Construct multipole expansions. For each childless box, build the pth-order multipole expansion for the field it generates. (This is done using recurrence relations, described in Section 1.4.)

Step 2: Propagate multipole expansions. Shift the centers of multipole coefficents from each child box to its parent's center and sum. This uses the multipole-to-multipole translation operator discussed in (Lin, 2000) and constructs a multipole expansion for the field generated by all charges contained in the parent box.

Comment: In the non-periodic case, boxes at levels 1 and 2 do not have well-separated neighbors, so multipole expansions are not useful for boxes at those levels (see below) and the iteration above can stop at level 3. In the periodic case, however, level 2 boxes can interact with well-separated *images* of neighboring boxes, and the multipole expansion for the largest (level 1) box is needed for contraction with the "lattice interaction tensor" (see below). So, for the periodic FMM, the loop needs to go up to level 1.

This comment applies to the propagation of local expansions as well.

1.2.3 Phase III

This section describes the "top-down" phase (again, see Figure 1.2):

Step 1: Propagate local coefficients. Starting at level 3 for non-periodic FMM and level 1 for periodic FMM (see previous section for rationale) and going down: For each box `b`, use the multipole-to-local translation operator to convert multipole expansions from all well-separated boxes (`List2(b)`) into a local expansion about the center of `b`. Then, if `b` is not childless, shift the local expansion in `b` to the centers of each child of `b`. This passes far-field effects from distant boxes to `b`'s children.

Step 2: Nearby large boxes In the adaptive case, some boxes may be too close to use the multipole-to-local translation operator. These boxes are collected in List 4 (defined below). For each such box, one can do a case analysis: If `bb` is in `List4(b)`, for example, and `bb` contains more than p^2 particles, then we can simply construct the local expansion for the field generated by charges in `bb` about the center of `b` and sum this expansion with the far-field effects computed in step 1. In fact, this step may be combined with the previous step.

Otherwise, just compute forces directly because that would be faster. (See the discussion in (Cheng *et al.*, 1999).)

This is a good time to define some terms and talk about various neighbor lists.

Well-separated boxes The main difference between this code and that of Cheng *et. al.* is the definition of well-separated neighbors: While they can accurately perform multipole-to-local conversion for boxes that only have a one-box separation, our translation operator requires a two-box separation. As a result, my list definitions differ from theirs.

First, some terminology: Two boxes are *neighbors* if they share a boundary point, and are *colleagues* if they are neighbors and are the same size (that is, if they are on the same level). They are *second colleagues* if they share a colleague, and *well-separated* if they occupy the same level and are not second colleagues. Note that every box is its own colleague and neighbor, and every colleague is also a second colleague. (To avoid confusion, the term "second neighbor" will be left undefined and unused.)

Neighbor lists Every box `b` has the following lists (note that List 0 is only used to define and construct other lists):

List 0: A box `bb` is in `List0(b)` if and only if it satisfies `level(bb)` $\leq$ `level(b)` and is a childless *neighbor* of `b` or a childless *second colleague* of `b`. Note that if `bb` is in `List0(b)`, then it necessarily occupies a lower level and is hence larger than `b`. This makes the list fairly easy to build, and it is very useful for defining and building other lists.

List 1: A box `bb` is in `List1(b)` if `bb` is either a childless neighbor of `b` or a childless second colleague of `b`. Particles in these boxes interact directly. Note that `bb` is in `List1(b)` if and only if either `bb` is in `List0(b)` or `b` is in `List0(bb)`.

List 2: A box `bb` is in `List2(b)` if `bb`'s parent is a first or second colleague of `b`'s parent *and* `b` and `bb` are not themselves first or second colleagues (i.e. they are well-separated). These boxes interact by multipole-to-local conversion.

List 3: `bb` is in `List3(b)` if and only if `b` is in `List0(Parent(bb))` but not in `List0(bb)`. In other words, `bb` is in `List3(b)` if and only if `bb` is strictly smaller than `b` and is near `b` but not adjacent to it. Thus the multipole expansion generated by `bb` converges in `b`, and it can be safely evaluated. (But see Phase IV notes, below.)

This list is useless (and hence undefined) if `b` is not childless.

List 4: A box `bb` is in `List4(b)` if and only if `b` is in `List3(bb)`. `List4(b)` may be nonempty for boxes `b` that have children, but its elements are all childless. If `bb` is in `List4(b)`, then `bb` is a larger, non-adjacent nearby box, making evaluation of local expansion possible.

Figure 1.2.3 illustrates some of these neighbor lists.

1.2.4 Phase IV

There is now enough information to compute the force acting on each particle, as follows:

Step 1: Direct interaction Let `b` be a childless box. For each box `bb` in `List1(b)` (which includes `b` itself), compute the forces resulting from direct interactions between the charges in `b` and those in `bb`. Do this for nearly-empty (that is, containing fewer than p^2 particles) boxes in Lists 3 and 4 as well.

Step 2: Evaluate multipole expansions Let `b` denote a childless box again. For each box `bb` in `List3(b)` containing more than p^2 charges, evaluate the force acting on each charge in `b` using the multipole expansion of `bb`. (This is less work than direct interaction.)

Note that one may be tempted to use the multipole-to-local translation operator here to reduce the amount of work. But a closer error analysis (relevant theorems are found in (Cheng *et al.*, 1999)) reveals that such expansions are not guaranteed to converge, and in practice this strategy can produce numerical instabilities.

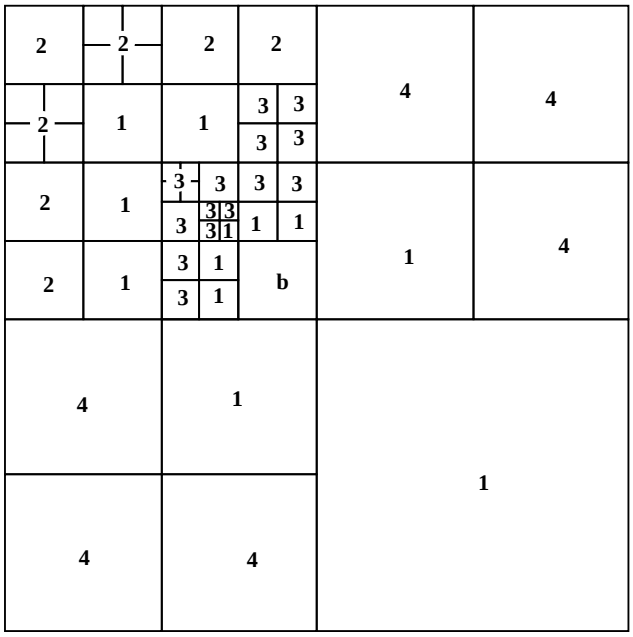

Figure 2: This figure illustrates the neighbors of box `b`. All boxes shown should be childless.

Step 3: Evaluate local expansions Each childless box `b` should have a local expansion of all far-field effects (computed recursively via List 2), as well as effects produced by large boxes nearby (through List 4 in the previous phase). Evaluate this expansion at each charge in `b`.

1.3 Periodic Boundary Conditions

My implementation of the periodic FMM follows Challacombe, White, and Head-Gordon exactly (Challacombe *et al.*, 1997). This approach allows only cubic cells, which may be a serious limitation for some calculations, but it is very efficient and relatively easy to implement.

First, some terminology: The (level 1) box containing all charges in a given simulation is known as the *simulation cell*, and we can think of periodic boundary conditions as specifying an infinite lattice of copies of this cell. Periodic copies are called *image cells* and exert an influence on charges in the simulation cell.

The periodic FMM is based on the following observation: Let $\{(q_i, \mathbf{r}_i)\}$ denote a system of N charges, let $\mathbf{r}_0$ denote the center of the simulation cell, and let $\hat{O}_{l,m}$ denote the multipole coefficients of the potential that these charges generate. Then, the electrostatic potential with periodic boundary conditions is given by

$$\sum_{\mathbf{n}} \sum_i \frac{q_i}{|\mathbf{r} - \mathbf{r}_i + \mathbf{n}|} = \tag{1}$$
$$\sum_{\mathbf{n}} \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \left[O_{l,m}(\mathbf{r} - \mathbf{r}_0) \sum_{j=0}^{\infty} \sum_{k=-j}^{j} M_{j+l,k+m}(\mathbf{n}) \cdot \hat{O}_{j,k} \right],$$

where the multipole-to-local translation operator (see (Lin, 2000)) was used to convert the multipole coefficients $\hat{O}_{l,m}$ generated by charges in the simulation cell into a local expansion of the field generated by *image* charges in image cells on the infinite periodic lattice.

Note that, as in the case of the non-periodic FMM, the multipole-to-local translation operator requires that cells be "well-separated" for convergence. Unlike the case for boxes, however, it turns out that a separation of one cell is sufficient in this case. Thus, the sum in Equation 1 actually only involves indices $\mathbf{n} = (n_1, n_2, n_3)$ for which $\max(|n_1|, |n_2|, |n_3|) > 1$, and neighboring image cells must interact "directly." (This is discussed below.) Challacombe, White, and Head-Gordon then go on provide a fast Ewald-like method for computing the coefficients

$$\sum_{\mathbf{n}} M_{l,m}(\mathbf{n}), \tag{2}$$

which I call the *lattice interaction matrix*.

This method can only handle cubic simulation cells. If the simulation requires adjusting cell dimensions dynamically, these adjustments must be uniform, so that the lattice interaction matrix can be scaled appropriately.

1.3.1 Modifications for periodic boundary calculations

The modificiations required are modest: First, the code needs to propagate multipole coefficients all the way up to level 1 (instead of only up to level 3), so that the multipole expansion for the entire simulation cell is obtained. (See Figure 3.) Next, this is contracted with a pre-computed lattice interaction matrix (see below) to generate an effective local expansion for image cell interactions. This can simply be propagated downwards and summed to local expansions from intra-simulation-cell far-field forces for smaller boxes in the simulation cell.

The slightly messy part involves neighboring image cells: As these cannot be included in the lattice interaction matrix, charges in the simulation cell need to interact "directly" with those in neighboring image boxes. This can be done by extending data structures so that boxes in neighboring image cells can be put into `List0` or `List2` (and hence into `List1`, `List3`, or `List4`) as if they are boxes in the simulation cell, with extended data structures that tag these image cells as such (so that translations and force evaluations are performed with appropriate shifts inserted).

1.3.2 Lattice interaction matrix

The following formulae, taken from (Challacombe *et al.*, 1997), lets one compute the lattice interaction matrix efficiently:

$$\sum_{\mathbf{n}} M_{l,m}(\mathbf{n}) = A_{l,m} + B_{l,m} \tag{3}$$

$$A_{l,m} = \sum_{|n_1|>1} \sum_{|n_2|>1} \sum_{|n_3|>1} (l-m)! P_{l,m}(\theta_{\mathbf{n}}) \cdot e^{im\phi_{\mathbf{n}}} \cdot G_l(\beta, n) \tag{4}$$

$$B_{l,m} = \frac{i^l \pi^{l-1/2}}{\Gamma(l+1/2)} \sum_{\mathbf{n}} n^{l-2} e^{-n^2\pi^2/\beta^2} \cdot (l-m)! \cdot P_{l,m}(\cos\theta_{\mathbf{n}}) e^{im\phi_{\mathbf{n}}} -$$

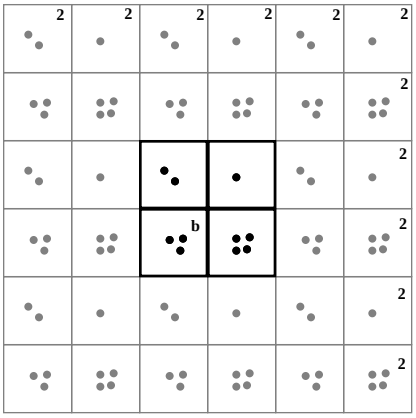

Figure 3: The simulation cell with neighboring (not well-separated) image cells: Periodic copies of the simulation cell are shaded, while the simulation cell and its charges are in solid black.. As the *cells* themselves are not well-separated, charges in these cells must interact at a lower level. Here, a box `b` is marked, along with images *boxes* that are well-separated from it. Box `b` is not intended to be childless, so its List 1 is (in constrast to Figure 1.2.3) undefined. Image cells not shown here contribute to the lattice interaction matrix.

$$\sum_{n_1=-1}^{1}\sum_{n_2=-1}^{1}\sum_{n_3=-1}^{1}(l-m)!\cdot P_{l,m}(\cos\theta_{\mathbf{n}})\cdot e^{im\phi_{\mathbf{n}}}\cdot F_l(\beta,n) \tag{5}$$

$$G_l(\beta,n) = \frac{\Gamma(l+1/2,\beta^2n^2)}{\Gamma(l+1/2)r^{l+1}} \tag{6}$$

$$F_l(\beta,n) = 1-G_l(\beta,n) \tag{7}$$

$$\Gamma(a+1,b) = a\Gamma(a,b)+b^ae^{-b} \tag{8}$$

$$\Gamma(1/2,b) = \sqrt{\pi}\,\mathrm{erfc}(\sqrt{b}) \tag{9}$$

Note that Equations 5 and 9, as they appear in (Challacombe *et al.*, 1997), are in error: (5) appears twice in the paper but is incorrect as Equation 17 of that paper and is correct in Equation A13, while (9) appeared with x instead of $\sqrt{x}$. The function $\Gamma(a)=\int_0^\infty t^{a-1}e^{-t}dt$ is the usual Gamma function, and $\Gamma(a,b)=\int_b^\infty t^{a-1}e^{-t}dt$ is the complement of the incomplete Gamma function $\gamma(a,b)=\int_0^b t^{a-1}e^{-t}dt$. ($\mathrm{erfc}(x)=1-\mathrm{erf}(x)$ is the complement of the usual error function $\mathrm{erf}(x)$.) In the above, β is a convergence factor and is usually set to $\sqrt{\pi}$ for optimal convergence rate.

I was not able to arrange the order of the computation so that the output of my program agrees with the lattice interaction matrix in (Challacombe *et al.*, 1997). So, I took the brute-force approach: The program `fmm_ndw` computes the lattice interaction matrix using quadruple precision arithmetic. This relies on SGI Fortran's `real*16` and `complex*32` data

types and associated intrinsic functions, and the program may not be portable. But, it works on at least one Silicon Graphics workstation. The program is fairly straightforward, with one little catch: To prevent loss of precision due to summing numbers with wildly different magnitudes, all the summands are computed first and sorted in ascending order before being summed.

1.4 Recurrence Relations: Details

This section documents facts about the internal representation and manipulation of multipole / local expansions.

1.4.1 Tables and index maps

Tables of multipole As noted in (Lin, 2000), multipole coefficients $\hat{O}_{l,m}$ satisfy

$$\hat{O}_{l,-m} = (-1)^m \cdot \overline{\hat{O}_{l,m}} \tag{10}$$

where overline denotes complex conjugation. Thus, we need only store $\hat{O}_{l,m}$ for $0 \leq m \leq l \leq p$, which are always stored in double precision complex arrays (`complex*16` arrays) in Fortran 77.

This presents a minor problem: The straightforward way to store $\hat{O}_{l,m}$ for $p = 10$, for example, is to simply declare a doubly-indexed array `A` of dimension $(11, 11)$. As memory is an issue on workstations, however, I opted to declare a linear complex array, say `A`, of size $\frac{(p+1)(p+2)}{2}$, and use a separate doubly-indexed integer array, usually called `ind`, whose indices range from 0 to p and contains appropriate indices into `A`. This works fairly well, but may not be as fast as possible. A parallel / optimized version of the code should probably remove this extra level of indirection, but any competent programmer can do that: There are many changes to be made, but they are all minor.

1.4.2 Recurrence relations for multipole expansions

This section contains the specific strategy used to compute multipole expansions, given an expansion center $\mathbf{r}_0$ and a charge at $\mathbf{r}_1$.

Let $\Delta\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_0$, and write $\Delta\mathbf{r}$ in spherical coordinates as (ρ, θ, ϕ).[7] Tables of multipole coefficients $O_{l,m} = O_{l,m}(\Delta\mathbf{r})$ are then computed as follows:

1. First, set $O_{0,0} = 1$ and $O_{1,0} = dz = \rho\cos\theta$.
2. Next, compute $O_{l,l} = \frac{1}{(2l)!!}\rho^l \cdot e^{-im\phi} \cdot \sin^l\theta$ iteratively for $l = 1, \ldots, p$.
3. Now that we have $O_{1,0}$ and $O_{l,l}, l = 0, \ldots, p$, we can compute $O_{l,0}$ for $2 \leq l \leq p$ using

$$O_{l,m} = \frac{2l-1}{l^2-m^2} \cdot \rho\cos\theta \cdot O_{l-1,m} - \frac{\rho^2}{l^2-m^2} \cdot O_{l-2,m} \tag{11}$$

with $m = 0$.

[7] To be absolutely pedantic: $\Delta\mathbf{r} = (\rho\sin\theta\cos\phi, \rho\sin\theta\sin\phi, \rho\cos\theta)$.

4. For $m = 1, \ldots, p-1$:

   (a) If $\sin\theta \geq |\cos\theta|$, then use

   $$O_{m+1,m} = \frac{e^{-i\phi}}{(2m+1)\sin\theta} \cdot (-2 \cdot \cos\theta \cdot O_{m+1,m-1} + \rho \cdot O_{m,m-1}) \tag{12}$$

   to compute $O_{m+1,m}$. We would then have $O_{m,m}$ (from Step 2) and $O_{m+1,m}$ from this step.

   (b) Else, if $\sin\theta < |\cos\theta|$, then use

   $$\begin{aligned} O_{m+1,m} &= -(m+1)\tan\theta \cdot e^{+i\phi} \cdot O_{m+1,m+1} + \\ &\quad \frac{\tan\theta}{2m+1} \cdot e^{-i\phi} \cdot O_{m+1,m-1} + \frac{m+1}{2m+1} \cdot \frac{\rho}{\cos\theta} \cdot O_{m,m} \end{aligned}$$

   to compute $O_{m+1,m}$. Again, we would have $O_{m,m}$ (from Step 2) and $O_{m+1,m}$ from this step.

5. Having computed $O_{m,m}$ and $O_{m+1,m}$, we can then use

   $$O_{l,m} = \frac{2l-1}{l^2-m^2} \cdot \rho\cos\theta \cdot O_{l-1,m} - \frac{\rho^2}{l^2-m^2} \cdot O_{l-2,m} \tag{13}$$

   to compute $O_{m+2,m}, \ldots, O_{p,m}$, and repeat the process.

1.4.3 Recurrence relations for local expansions

Again, let $\Delta\mathbf{r} = \mathbf{r}_1 - \mathbf{r}_0$ and $\Delta\mathbf{r} = (\rho, \theta, \phi)$ in spherical coordinates. Tables of local coefficients $M_{l,m} = M_{l,m}(\Delta\mathbf{r})$ are then computed as follows:

1. First, set $M_{0,0} = 1/\rho$ and $M_{1,0} = \cos\theta/\rho^2$.

2. Next, compute $M_{l,l} = \frac{(2l-1)!!}{\rho^{l+1}} \cdot e^{im\phi} \cdot \sin^l\theta$ iteratively for $l = 1, \ldots, p$.

3. Now that we have $M_{1,0}$ and $M_{l,l}, l = 0, \ldots, p$, we can compute $M_{l,0}$ for $2 \leq l \leq p$ using

   $$M_{l,m} = \frac{(2l-1)\cos\theta}{\rho} \cdot M_{l-1,m} - \frac{l^2-m^2}{\rho^2} \cdot M_{l-2,m} \tag{14}$$

   with $m = 0$.

4. For $m = 1, \ldots, p-1$:

   (a) If $\sin\theta \geq |\cos\theta|$, then use

   $$M_{m+1,m} = \frac{e^{i\phi}}{\sin\theta} \cdot \left(-\cos\theta \cdot M_{m+1,m-1} + \frac{2m}{\rho} \cdot M_{m,m-1} \right) \tag{15}$$

   to compute $M_{m+1,m}$. We would then have $M_{m,m}$ (from Step 2) and $M_{m+1,m}$ from this step.

(b) Else, if $\sin\theta < |\cos\theta|$, then use

$$\begin{aligned} M_{m+1,m} &= -\frac{1}{2}\tan\theta \cdot e^{-i\phi} \cdot M_{m+1,m+1} + \\ &\quad \frac{1}{2}\tan\theta \cdot e^{i\phi} \cdot M_{m+1,m-1} + \frac{m+1}{\rho\cos\theta} \cdot M_{m,m} \end{aligned} \tag{16}$$

to compute $M_{m+1,m}$. Again, we would have $M_{m,m}$ (from Step 2) and $O_{m+1,m}$ from this step.

5. Having computed $M_{m,m}$ and $M_{m+1,m}$, we can then use

$$M_{l,m} = \frac{(2l-1)\cos\theta}{\rho} \cdot M_{l-1,m} - \frac{l^2-m^2}{\rho^2} \cdot M_{l-2,m} \tag{17}$$

to compute $M_{m+2,m}, \ldots, M_{p,m}$, and repeat the process.

1.4.4 Computing gradients

The recurrence relations above can be used to compute multipole coefficients $\hat{O}_{l,m} = \sum_i q_i \cdot O_{l,m}(\mathbf{r}_i - \mathbf{r}_0)$ and $\hat{M}_{l,m} = \sum_i q_i \cdot M_{l,m}(\mathbf{r}_i - \mathbf{r}_0)$ for clusters of charges $\{(q_i, \mathbf{r}_i)\}$. The corresponding electrostatic potential U is given by

$$\begin{aligned} U(\mathbf{r}) &= \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \hat{M}_{l,m} O_{l,m}(\mathbf{r}-\mathbf{r}_0) \\ &\quad \sum_{l=0}^{\infty}\sum_{m=-l}^{l} M_{l,m}(\mathbf{r}-\mathbf{r}_0)\hat{O}_{l,m}. \end{aligned} \tag{18}$$

To compute the gradient $\nabla U(\mathbf{r})$, one needs to instead compute

$$\begin{aligned} \nabla U(\mathbf{r}) &= \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \hat{M}_{l,m} \nabla O_{l,m}(\mathbf{r}-\mathbf{r}_0) \\ &\quad \sum_{l=0}^{\infty}\sum_{m=-l}^{l} \nabla M_{l,m}(\mathbf{r}-\mathbf{r}_0)\hat{O}_{l,m}. \end{aligned} \tag{19}$$

The coefficients $\nabla O_{l,m}$ and $\nabla M_{l,m}$ can be computed by differentiating the recurrence relations from the previous sections with respect to r, θ, and ϕ, and then converting the gradient from spherical to cartesian coordinates. This is a straightforward calculus exercise and is omitted here.

Numerical stability It is possible to arrange the recurrence relation in Steps 3 and 5 in both multipole and local computations above to make it numerically stable.(Arfken, 1985) *This was not done and the numerical stability of the other recurrence relations have not been checked.* Empirically, this does not cause problems with uniform distributions of particles, but perhaps someone better versed in numerical analysis than I should check the numerical stability of these recurrence relations and rearrange the computation.

1.4.5 Faster translation operators

The translation operators in this code follow (Lin, 2000), which are simple $O(p^3)$ algorithms. Please look there for an explanation of recurrence-based translation operators.

Multipole-to-multipole translation The multipole-to-multipole translation operator requires three auxiliary arrays. This is because, in the notation of (Lin, 2000), it requires the partial sums $R^j_{l,0}$, $R^j_{l,l}$, and $R^j_{l,l-1}$ for $l = 0, \ldots, p$. Each of these arrays have $O(p^2)$ elements and require $O(p)$ operations per element to compute. It is then straightforward (again, see (Lin, 2000)) to use these elements to compute all the partial sums $R^j_{l,m}$ and sum them over j to get the translated multipole coefficients.

Note that the code actually loops over j on the outside loop, so for each j we have l running from j to p: This enforces the constraint that $R^j_{l,m}$ exists only for $j \leq l$.

Local-to-local translation Unlike the multipole-to-multipole operator, the local-to-local operator only requires one auxiliary array. This is because, by definition, we have $S^j_{j,m} = \hat{M}_{j,m}$ for $m = 0, \ldots, p$, so we can just pull that from the original array rather than having to store it. The only partial sums we need to compute to start the recurrence relations, then, are $S^j_{l,0}$ for $0 \leq l \leq j \leq p$.

Again, the code reorders the sum by looping over j on the outer loop, then lets l run from $j - 1$ down to 1: This ensures that $l \leq j$ holds.

Multipole-to-local translation As mentioned in (Lin, 2000), the multipole-to-local translation operator uses recurrence relations with $\sin\theta$ in the denominator, which can cause problems when $\sin\theta = 0$. But in such cases, $\theta = 0$ or $\theta = \pi$ holds, so that we would be translating coefficients along the $\hat{\mathbf{z}}$-axis. So, the code starts by checking if $\sin\theta = 0$ holds, and if it does, to simply use the $O(p^3)$ algorithm for vertical translations (see (White and Head-Gordon, 1996)). Otherwise, it uses the recurrence based algorithm described in (Lin, 2000).

The computation of translated local coefficients from partial sums $T^j_{l,m}$ requires two auxiliary arrays: They must hold $T^j_{l,0}$ and $T^j_{l,1}$ for $j, l = 0, \ldots, p$. The code is fairly straightforward.

The code to compute the vertical translation operator is also straightforward, but it tries to compute various constants involving factorials and powers of -1 iteratively, so it may look a bit confusing at first.

Numerical stability In numerical tests with uniformly distributed charges, the recurrences have appeared numerically stable for $N \approx 1000$ and $p \approx 40$. No rigorous stability proof has been done, but a good numerical analyst should be able to figure this out.

Array bound checking As mentioned in (Lin, 2000), the recurrences for $R^j_{l,m}$ (used in multipole-to-multipole translation) and for $S^j_{l,m}$ (used in local-to-local translation) actually refer to multipole coefficients $O_{l',m'}$ for which $|m'| > l'$, which are 0 by definition. The code calls a procedure to ensure that this happens, which is robust and easier to read, but slow.

An optimized code should unroll the loops to only sum over nonzero multipole coefficients.

Matrix element caching All translation operators currently incur a $O(p^2)$ overhead by calling constructors for multipole or local coefficients, to construct the $O_{l,m}(\Delta\mathbf{r})$ or $M_{l,m}(\Delta\mathbf{r})$ coefficients necessary for performing the translation. But, by its very nature, the multipole algorithm actually does translations along the same directions very often. It is therefore possible to rearrange the computation so that these matrix elements are not recomputed. This has not been done, and may not be easy to do because of the data structures used to store neighbor lists and the way the multipole calculation is organized, but anyone wishing to use this code effectively should beware this not-so-hidden extra cost.[8]

1.5 FMM: Details

1.5.1 Data structures

Neighbor lists are implemented as singly-linked lists. As Fortran 77 does not provide pointers or dynamical memory allocation, this must be simulated using Fortran arrays, following Cormen, Leiserson, and Rivest (Cormen *et al.*, 1990): A large integer array `A` is first allocated,[9] with dimension `(2,llsize)`, where `llsize` is a large integral constant. For each entry in the array, the entry `A(1,i)` holds the datum (usually a pointer to a box – see below) and `A(2,i)` holds a pointer to the next entry of `A` in the linked list. Or, it may hold a predefined constant, `null`, if it is the last cell in the list.[10]

Similarly, it is useful to have "objects," which in my code is implemented by having separate arrays represent different data fields for a large list of objects of the same type. (See (Cormen *et al.*, 1990).)

Very large arrays are allocated so that there is enough space for each box in a full tree of some given depth `maxlevel`.[11] This is a waste of space, but easier to work with. Every box has two kinds of labels, an *index* and a set of *coordinates*. An index is just an integer pointer into these really big arrays that hold various fields of (simulated) data structures. Coordinates are sets of four integers, `(level,i,j,k)`, where `level` is the level the box occupies, and $0 \le \mathtt{i,j,k} \le 2^{\mathtt{level}-1} - 1$. A simple function computes the index from the coordinate; the inverse function is not needed because each box has fields to store its own coordinates.

[8] I hope that the parallel version is sufficiently fast that this optimization is necessary, because it may not be trivial to implement.

[9] In the code, for random reasons, this large array is called `l37` instead. Sorry if this bothers you: Just change it to `l42` or whatever else strikes your fancy.

[10] For entirely arbitrary reasons, `null` is set to -123456789: A sufficiently large negative number that attempts to use this as an array index should crash the program.

[11] As the number of boxes in a full tree of depth `maxlevel` is $\frac{1}{7}(8^{\mathtt{level}} - 1)$, the storage required grows very quickly with `maxlevel`. Fortunately, in practical problems with $N \approx 10^6$, `maxlevel` need not be much larger than 5 or 6.

1.5.2 Correctness of list structure

It is not entirely obvious that the list definitions given before are entirely correct. One can indeed attempt to construct a correctness proof for these lists, but such a pedantic exercise is not very helpful: It can be every bit as convoluted as the list definitions themselves, and in any case the ultimate test of correctness is whether these lists allow us to compute electrostatic potentials and forces correctly or not, which they do.

1.6 Efficiency

There is lots of room for improvement in this area because the code was written to be slow. Not just accidentally slow, but deliberately slow.

1.6.1 Array index maps

Just getting rid of index maps gets rid of a lot of memory references but introduces some arithmetic operations. Whether or not this tradeoff speeds up the program significantly depends on the architecture, available memory, and whether one rewrites all loops so that inner loops access contiguous array entries: Just getting rid of indices without rewriting loops to sum over inner loops first (as Fortran prefers) doesn't improve performance very much.

1.6.2 Data structures

I think these linked lists and stuff work very well and very quickly. What sucks about it is that it makes exploiting symmetries to reuse translation matrices rather difficult. I do not know of a good way around that, and I hope this is not a big issue for the parallelized version.

1.6.3 Optimal parameters and tree balancing

The running time of the algorithm depends very sensitively on the depth of the tree and the number of particles in childless boxes, which in turn depends very sensitively on the parameter `maxcount`. Following the analysis in (Cheng *et al.*, 1999) and adjusting some constants to reflect the running time of our algorithm, it looks like a $O(p^{3/2})$ scaling for `maxcount` is about right. What constant factor to put in front of $p^{3/2}$ is a matter of experimentation.

1.6.4 The cost of adaptivity

Adaptive code is actually fast: most of the time is spent doing translation operators. List and tree construction, while complicated on paper, are a breeze for the program. Of course, the adaptive code is much more complicated than non-adaptive code, and maybe the non-adaptive code is easier to optimize. It depends on the application.

It was suggested ((Pohorille, n.d.)) that adaptivity should be an option: Maybe one can construct the tree and use it in a multipole calculation using separate subroutines, so that

the tree need not be reconstructed at every time step. This may be nice, but, depending on how parallel tree construction is done, it may not improve performance very much. The code, as it stands, requires some work to make adaptivity optional.

1.6.5 Exploiting symmetry

See section on "data structures," above.

1.7 Limits

There are some limits to what this code can do.

1.7.1 Memory allocation

Lots of big arrays are pre-allocated. Their sizes are fixed by constants. That is bad news on a workstation, tolerable on large parallel machines with lots of real estate.

1.7.2 Numerical stability

Multipole and local coefficient construction, as well as translation operators, involve using recurrence relations whose status as stable numerical algorithms is unknown (but knowable).

There's also a little problem with the routines that convert the gradient from spherical coordinates to cartesian coordinates: They do not check for division-by-zero errors.

1.7.3 Periodic boundary conditions

The periodic fast multipole method of Challacombe, White, and Head-Gordon (Challacombe *et al.*, 1997) is nice and fast, but limits what one can do in terms of changing the size of the box during a computation: If the simulation cell is scaled uniformly, it is easy to figure out how to rescale the effective interaction matrix accordingly. But, if there is a need to scale each axis separately, as in constant pressure tensor calculations, then we (actually, you) are out of luck.

2 Establishing a NASA Resource in Bioinformatics

(This project was carried out by Kate Parmer, an undergraduate student from U.C. San Diego with Andrew Pohorille and Michael New)

One of the goals of the bioinformatics resource is to provide researchers with unique, and custom tools to apply to public, or private genomic data, and one type of analysis critical to understanding homology is a multiple alignment, whereby similar sequences are aligned, and presented in a tabular format.

A multiple alignment is usually viewed manually, and features are noted by direct inspection. However, it is sometimes necessary to quantitate how good an alignment is. In this case, a calculation of relative entropy on the individual columns can provide information

on how well conserved a particular region of an alignment is. This can provide insight into important functional domains in proteins, such as enzyme active sites, and drug binding sites.

A program was written to read in a multiple alignment, regularize the character probabilities, and perform a relative entropy, and mutual information calculation. This involved calculating the probability distributions for the individual columns of the alignment, performing Dirichlet regularization (in the case of protein sequences) or pseudocounts (for nucleic acid sequences), and then performing the calculations on these "re-normalized" probability distributions. This normalization is required since for alignments with only a few sequences, some of the character probabilities in a column may be zero, due to under-representation of that character in a particular column. In addition, mutual information between all columns is calculated. This can give insight into correlation between columns in the alignment.

This work will be implemented on the NASA bioinformatics resource website in the near future.

3 Protobiological Chemistries using MolNets

(This project was carried out by Kenneth Kang, an undergraduate student from Stanford with Silvano Colombano)

MolNets provide a model protochemistry, which allows us to study models of chemical evolution before the genome. They are neural networks which emulation molecular chemistry, where the neurons are analogous to atoms and connections are analogous to chemical bonds. MolNets can "react" with one another to form new structures. For prebiotic chemistry, we can associate "atoms" with amino acids and "molecules" with proteins. The challenge is then to devise rules and parameters that make physical sense without imposing a desired outcome on the system. In particular, we want to allow for the possibility of emergent behavior.

At the present time, we have implemented a set of generalized, bond-centric rules to generate artificial chemistry. We have implemented a common code for comparing, saving and loading molecules. Reactions bond the reactants together, or bonds break and the products are separated. The tools that we have developed to do this include a controlled environment which examines each reaction and searches for metabolic reaction cycles. In Figure 4a, we show the generalized reaction mechanism, and Figure 4b show a hypothetical reaction cycle. Figure 5 shows a MolNet polymerization reaction. The reactant on the left can react with itself to form a dimer. The reactant can be added to the dimer to form a trimer, and so on indefinitely. In the MolNet artificial chemistry, 12 monomers with fewer than 8 atoms were found.

Just like we see in the "inherited efficiencies" model, populations of MolNets can show a shift towards larger average size and greater complexity of interaction. We believe that they can represent an experimental substrate for studying the type of increasing complexity that was a necessary precursor for the evolution of primitive organisms. Models of molecular self-organization might also contribute to nanotechnology and production of new materials and nanoscale sensors and devices.

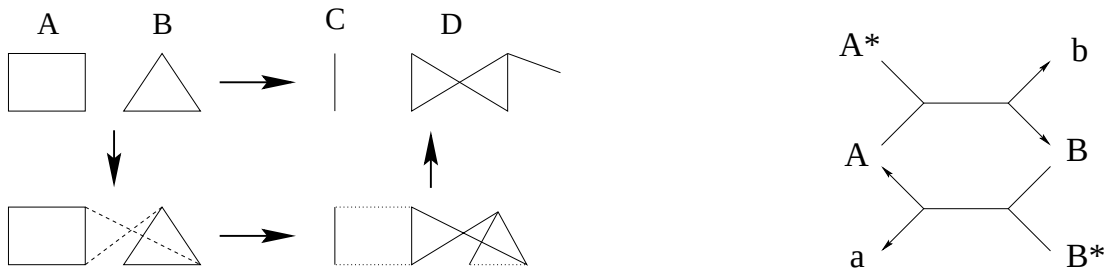

Figure 4: (a) Generalized Reaction Mechanism. The dashed lines indicate bonds being formed, the dotted lines indicate bonds being broken; (b) Hypothetical reaction cycle.

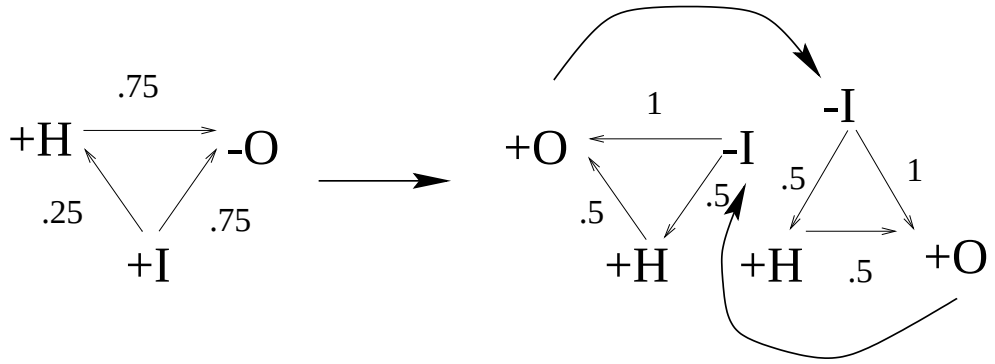

Figure 5: MolNet polymerization reaction.

4 Worlds in Collision – Dynamical Interactions within Three-Planet Systems

(This project was the work of Justin Lacy, an undergraduate student in Physics from George Mason University, with Greg Laughlin and John Chambers)

The census of 80-odd extrasolar planets discovered thus far contains many objects with highly eccentric orbits and/or small orbital radii. It is believed that this bizarre collection of orbits has been produced by "dynamical" instabilities involving two or more planets modifying each other's orbits via their mutual gravitational pulls. The main thrust of Justin Lacy's summer project was to make a statistical examination of the kinds of dynamical instabilities that occur within systems of three Jupiter-mass planets started on initially circular orbits.

Justin performed several thousand integrations of model three-member which resemble the Jupiter-Saturn-Uranus trio. Our own Solar System is extremely stable, due to the low masses of Saturn and Uranus in comparison to Jupiter. In each simulation, however, the planet triples were each endowed with a Jovian mass, and the evolution of the orbital dynamics was investigated over timescales ranging from 10–100 million years. This represented

an enormous computational effort, which was performed with a distributed parallel system.

In many cases, the model systems experienced severely unstable trajectories, leading to collisions, ejections, and large orbital migration among the planets. Our main conclusion is that *planetary scattering from Jovian distances cannot adequately account for the observed distribution of orbits.* Justin's results convincingly indicated that it is difficult or impossible to account for the observed aggregate of extrasolar planets with a scenario that involves multiple planets on initially circular orbits which form beyond the so-called "snow line" (located at about 5 astronomical units — the orbital radius of present-day Jupiter). Justin's results strongly suggest that the observed extrasolar planets either formed well inside the snow-line, or underwent significant orbital migration before interacting dynamically.

His results are important because they show that the formation scenario for the known extrasolar planets **must** have been qualitatively different from the formation scenario that led to the solar system.

References

Arfken, George B. (1985). "Mathematical Methods for Physicists, " second ed. Academic Press, New York.

Challacombe, Matt, White, Chris, and Head-Gordon, Martin. (1997). *Journal of Chemical Physics* **107**.

Cheng, H., Greengard, L., and Rokhlin, V. (1999). *Journal of Computational Physics*.

Cormen, R., Leiserson, C., and Rivest, R. (1990). "Introduction to Algorithms." MIT Press, Cambridge, Massachusetts.

Lin, Kevin K. (2000). Simple Approach to $O(p^3)$ Scaling in the Fast Multipole Method. Unpublished.

Pohorille, Andrew. Private communication.

White, Christopher A., and Head-Gordon, Martin. (1994). *Journal of Chemical Physics* **101**.

White, Christopher A., and Head-Gordon, Martin. (1996). *Journal of Chemical Physics* **105**.